**Instructor: Dr. S. Hughes**

**Due date: Tuesday, Jan 23nd (midnight)**

# Extreme Nonlinear Optics: Dynamics of Coupled ODEs

**Keywords**: Euler, RK4, Basic `matplotlib` Graphics, FFTs, `odeint` (from `scipy`), Optical Bloch Equations, Rotating-Wave Approximation, Beyond the Rotating-Wave Approximation, Rabi Flopping, Area Theorem.

**Task**: go through all the questions and coding exercises below and then write up a small physics report in the LaTeX template provided: Simple Report Template

**Marks**: Codes and Results (20), Results (10). Total: (30)

**Reminder:** All codes must run under Python 3.9.x. and the Spyder 5.x IDE (check this before submitting)

**Some Background:**

**From Rubin Landau's web page on Computational Physics (hyperlinks):**
(1) Numerical Solution to Differential Equations (basic, so not needed for most of you): video (31 mins)
(2) ODE Algorithms: video (36 mins)
**From Wikipedia:**
Rabi cycle and Rabi flopping
**Three Relevant Research Papers (all of which were published in *Physical Review Letters*), and you can probably find lots more on this general topic:**
Breakdown of the Area Theorem: Carrier-Wave Rabi Flopping of Femtosecond Optical Pulses
Signatures of Carrier-Wave Rabi Flopping in GaAs
Carrier-Wave Rabi-Flopping Signatures in High-Order Harmonic Generation for Alkali Atoms

## Question 1

In the class notes, we introduced the Optical Bloch equations (OBEs) in various forms. The simplest form was for CW excitation (*continuous wave*, monochromatic harmonic wave) in the RWA (rotating-wave approximation) and the interaction picture:

$$\boxed{\begin{aligned} \frac{du}{dt} &= -i\Delta_{0L}u + i\frac{\Omega_0}{2}(2n_e - 1) \quad (1)\\ \frac{dn_e}{dt} &= -\Omega_0 \mathrm{Im}[u], \quad (2) \end{aligned}}$$

where $u{=}\rho_{eg}$ is the (slowly varying) *complex* coherence and $n_e{=}\rho_{ee}$ is the population density of the excited state of a two-level system (TLS), or "quantum bit" (qubit).

(a) Begin with the example "sample bad code" onQ (ODESolver_Bad1.0.py) (deliberate!). The basic core of the code has two features:

(i) The OBEs to send to an ODE solver:

```
"ODEs - with simple CW excitation"
def derivs(y,t): # derivatives function
    dy=np.zeros((len(y)))
    #dy = [0] * len(y) # could also use lists here which can be faster if
                       # using non-vectorized ODEs "
    dy[0] = 0.
    dy[1] = Omega/2*(2.*y[2]-1.)
    dy[2] = -Omega*y[1]
    return dy
```

(ii) A *vectorized* ODE solver using the simple Euler method:

```
def EulerForward(f,y,t,h): # Vectorized forward Euler (so no need to loop)
# asarray converts to np array - so you can pass lists or numpy arrays
    k1 = h*np.asarray(f(y,t))
    y=y+k1
    return y
```

[Note: You can delete the `np.asarray` if using `numpy` arrays, which will then be very slightly faster (probably). If you usually use `numpy` arrays (and you should!), you can remove these.]

For on-resonance excitation ($\Delta_{0L} = 0$), the sample code solves for five complete Rabi oscillations in time, and compares the numerical solution with the analytical solution for the excited state population: $n_e(t) = \cos^2(\Omega_0 t)$, when $n_e(0) = 0$. We will use scaled time units $t_s = t/t_p$, so the scaled frequencies are then $\Omega^s = \Omega t_p$ (same for $\omega_L^s = \Omega_L t_p$), etc.

In normalized time units, we use the Rabi frequency $\Omega_0 t_p = 2\pi$, and so the final (scaled) time is $tend = 5$. ($n2\pi$ gives $n$ full cycles). The time step in the example code is 0.001 ("ridiculously small"), and as you can see the solution has still not converged. This code uses 1000 points per period, which is well within the general rule of thumb of having 100 or 1000 points over the characteristic time (or length) scale, such as the Rabi period here. This "problem" is well known in scientific computing, and is why the usual first-try approach we prefer is the Runge-Kutta fourth-order method (aka: "RK4"). Every decent physicist needs to know how to write up some RK4 code ☺, so we will get this out of the way with the exercises below.

(b) Implement an RK4 ODE solver and compare the solution with the "toy" Euler solution. Plot all three solutions (analytic, Euler, RK4), in a way that one can see (distinguish) the three solutions.

(c) Reduce the step size to the "rule of thumb" mentioned above, so $h$=0.01 (see also Landau's video). Now plot the RK4 solution versus the analytical solution (this problem is commonly encountered in showing similar results, and requires attention to the graphical outputs for your report, and line types).

(d) Substantially improve the graphics and potting (defaults in most languages are terrible!) to create two nice graphs that you can include in your LaTeX report (see template). Label everything clearly with fonts and lines that show all the features, lines, and satisfy the criteria of a good figure in a Physics Journal paper, such as *Physical Review A/B*. Keep in this style/template moving forward, so you do not have to keep working with bad figures (and so I do not have to keep reminding you and taking lots of points off for bad graphs that hurt my eyes ☹). Indeed, it is worth spending the time to create various *templates* for figures (single figure, 2 panel, 4 panel, simple animation, etc, though you can also do this when the time comes.)

****General graph rule** - PLEASE READ CAREFULLY AND ALWAYS REMEMBER - no fonts on the graphs should be smaller than the figure caption fonts! It's that simple.**

## Question 2

Now that you have a working ODE solver that is efficient and has been numerically checked against an analytic solution (with hopefully excellent agreement), we want to explore the OBEs with a time-dependent pulse (e.g., a pulsed laser). For this question, we will use the standard RWA equations:

$$\boxed{\begin{aligned} \frac{du}{dt} &= -\gamma_d u - i\Delta_{0L} u + i\frac{\tilde{\Omega}(t)}{2}(2n_e - 1) \quad (3)\\ \frac{dn_e}{dt} &= -\tilde{\Omega}(t)\mathrm{Im}[u], \quad (4) \end{aligned}}$$

and consider a Gaussian pulse:

$$\tilde{\Omega}(t) \to \tilde{\Omega}_{\mathrm{Gauss}}(t) = \Omega_0 \exp\left(-t^2/t_p^2\right), \quad (5)$$

where $t_p$ is the pulse width.

Initially, we consider on-resonance excitation and no dephasing (as before), so $\Delta_{0L} = 0$ and $\gamma_d = 0$. In units of scaled time $t \to t/t_p$, consider a pulse area of $2\pi$, and confirm numerically that a perfect Rabi flop occurs, namely $n_e$ goes from 0 to 1 and exactly back to zero at the end of the pulse. Run your simulation for a total time of $tend = 10$ (so 10 pulse durations), and add a sensible offset to the pulse (e.g., $5t_p$, or 5 in normalized units, is appropriate, ensuring that the pulse is turned on well away from the center of the pulse, so it starts from almost zero.)

## Question 3

Next we will solve the full Rabi problem with no rotating wave approximation (exciting stuff!). We are going into a domain where no analytical solution exists. We will consider the full-wave Bloch equations:

$$\boxed{\begin{aligned} \frac{du}{dt} &= -\gamma_d u - i\omega_0 u + i\Omega(t)(2n_e - 1) \quad (6)\\ \frac{dn_e}{dt} &= -2\Omega(t)\mathrm{Im}[u], \quad (7) \end{aligned}}$$

where $u$ is now quickly-varying (you may have to adjust your time step appropriately), and $\Omega(t)$ is a full-wave Rabi field:

$$\Omega(t) \to \Omega_{\mathrm{Gauss}}(t) = \Omega_0 \exp\left(-t^2/t_p^2\right) \sin(\omega_L t + \phi), \quad (8)$$

and initially choose $\phi = 0$ and $\gamma_d = 0$. We will stay on resonance, so $\omega_0 = \omega_L$, but increase your simulation time to $50t_p$ (it will help with the Fourier transforms below).

(a) Using a nominal $2\pi$ pulse (as defined from the RWA envelope solution – you can work this out analytically, which will define $\Omega_0^{2\pi}$), investigate the features of $u(t)$ and $n_e(t)$ when $\omega_L = \Omega_0^{2\pi}, 2\Omega_0^{2\pi}, 8\Omega_0^{2\pi}$. Show the numerical results graphically and compare and contrast with the RWA solution, which should show the "Area Theorem." Investigate if changing the phase to $\phi = \pi/2$ makes any difference on your findings, and comment on your findings. Can the RWA models account for the phase of the drive, and would it make any difference to the solution?

(b) Next, fix $\omega_L = 2\Omega_0^{2\pi}$ and show the results for nominal pulse Areas of $\pi/2, 4\pi, 16\pi$. Comment on your findings.

(c) In ultrashort pulse experiments, it is common to measure the frequency content of the transmitted or reflected pulse, through the power spectrum, $|E(\omega)|$. In a thin-sample approximation (the medium that contains the TLS material), we can relate to this quantity by studying the power spectrum of the polarization, which is related to $|u(\omega)|$.

Using a polarization decay rate of $\gamma_d = 0.2/t_p$, plot the polarization power spectrum in normalized units of $\omega/\omega_L$ for $|u(\omega)|$ for the three cases studies in (b). Check your transient has decayed to approximately zero, which should be the case when you add in $\gamma_d$ for all cases. You can use the `numpy` functions for your FFTs (fast Fourier transforms), including, e.g., `np.fft.fft`, `np.fft.fftfreq`, and `np.fft.fftshift`. You should become familiar with these functions as they are typically used quite often. In sensible normalized units, plot the solution $\omega/\omega_L$ ranging from 0 to 6 (so 6 times the laser frequency). Comment on the features you obtain.

Also plot the power spectrum of the excitation pulse, on the same graph, namely $|\Omega(\omega)|$ in sensible normalized units (e.g., a peak at 0.95 or 1), and check it peaks at $\omega/\omega_L = 1$. If it does not, you have a units problem - this will need fixed otherwise. This quantity is the input spectrum, and it is useful to see how it differs to the output spectrum.

## Question 4

- Finally, using some of your final example code above (e.g., in 3(a)), compare your code results and run times (give the results from a python timer, e.g., using `import timeit`, `start = timeit.default_timer()`, `stop = timeit.default_timer()`) versus the python `odeint` routine available from `SciPy`. You only need to test for one full-wave example, such as one of the cases in 3(b).

  Some pseudo-code for `odeint` is given below:

```
# read in ode solver
from scipy.integrate import odeint
...
y0=[0.,0.,0.] # initial condition
# options can also be includes here for errors and additional step sizes, such
                                          as mxstep=20 would allow up to 20
                                         sub divisions of your chosen time
                                         array step
y=odeint(OBEsFull,y0,t)
pop = y[:,2]
```

  Note `odeint` cannot deal with complex numbers (not a big deal for our fairly simple examples in this assignment, as we can work out the real and imaginary components by hand), and you can easily separate $u$ into real and imag parts. However, odeintw can, but we do not need to look into that right now.

  Note also than alternatively you could use one of the `solve_ivp` routines, which are newer and also from `scipy`, but they have a slightly different syntax and tend to be a bit slower (extra overhead from trying to mimic Matlab-style ODE suites); the default method for the solver is RK45 (a 5th order Runge-Kutta-Fehlberg method), which you can also call explicitly by passing: `method='RK45'`). In these `scipy` routines, you can also explicitly give the absolute and relative errors, which can have certain advantages (as they will usually adjust the step size accordingly – "adaptive step", which is beyond what we will do just now). These can also be used to benchmark the accuracy of your own solvers, though we can always write an adaptive step to do that as well. We come back to these ODE solvers for later assignments, and you will probably be using your RK4 for the rest of your life, so add some sensible comments so you (and others) know whet is going on.

**THIS IS WORTH REPEATING: **General graph rule** - PLEASE READ CAREFULLY AND ALWAYS REMEMBER - no fonts on the graphs should be smaller than the figure caption fonts! It's that simple.**